

Linux 資源管理に関する取り組みの紹介

早稲田大学理工学研究科
菅谷みどり

中島研究室

<https://www.dcl.info.waseda.ac.jp/home/index.html>

TechnicalJamboree4
2005/09/30

内容

1. 背景
2. 資源管理に関する Linux の課題
3. 取り組み事例
 - 組み込み Linux 向け資源管理システム(CABI)
 - Emblix WG でのプロセスQoSに関する標準化の取り組み
4. まとめ

背景

- ユビキタスコンピューティング環境
 - 無線通信機能, 計算機能, 小型集積技術の発展
- 組み込みシステム
 - 適用分野の拡大
 - 情報家電, 携帯電話, カーナビ, デジタル TV, PDA, etc
 - 大規模化, 複雑化
 - GUIの高度化, ネットワーク化
 - 開発期間の短縮化, コストダウンへの要求
 - 汎用OSの利用
 - Linux の利用
 - CELF, 家電用パッチの公開 (2004,06)
- 多くの製品で Linux が利用されるようになった
 - 応答性, バッテリー管理, ブートタイムなど様々な問題が解決されつつある
 - CELFによる標準化
 - Embedded Linux の認知度の高まり
 - コミュニティへのパッチの提供
 - 残されている課題
 - 資源管理
 - セキュリティ, 信頼性...

組み込みLinux製品



PDA's and handheld computers



Linux-based mobile phones and smartphones



Linux-powered robots



Audio/video entertainment devices

2005/09/30

TechnicaUmboree4

3

Linuxの資源管理

- 資源管理対象
 - メモリ, I/O は早いスピードで改善
 - メモリ
 - NUMA, ゾーン, ページ, キャッシュ, DMA, MMU
 - I/O
 - Multi page block I/O
 - I/O Scheduling
 - AIO (Asynchronous I/O)
 - CPU に関しては...
 - 応答性の問題
 - プリエンプティブルカーネルによる応答性の向上
 - O(1) スケジューラによる jitter の縮小
 - » しかし, 優先度が低いプロセスの応答性は未解決
 - 資源占有の問題が残る
 - セキュリティとも絡む重要な問題

2005/09/30

TechnicaUmboree4

4

CPU資源管理に対する組み込み Linux の課題

1. 資源占有によるシステム信頼性の低下への対応

- ネットワーク接続
 - ダウンロードされた未検証プログラムの実行
 - 資源占有の危険
 - » 安全性の低いプログラムによる資源占有を防ぐには？

2. 応答性の向上

- Linux コミュニティの尽力によりLatency の縮小は進行
 - クリティカルセクションの縮小, 全体周期の細粒度化
 - » しかし, GUIの応答性の改善は不十分

3. 多様なアプリケーションの動作

- Linux は目的が異なる二つのスケジューリングを考慮する必要がある
 - リアルタイムプロセス + タイムシェアリングプロセスの動作
 - » リアルタイム = 最悪値保証
 - » タイムシェアリング = 平均性能の向上

2005/09/30

TechnicalJamboree4

5

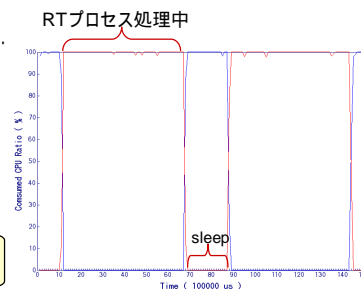
例えば, 制御せずにリアルタイムプロセスを動作させると...

• 優先度が高いプロセスの資源占有

- Linux は優先度が高いプロセスを走らせ続ける
 - 適切な管理(設定)を行わないと, システム全ての資源を利用してしまふ.
 - リアルタイムプロセスの動作中, タイムシェアリングプロセスはCPUを得られない.
 - 適切な優先度, 周期時間管理が必要
 - しかし,
 - システムの複雑化, 大規模化
 - 多様なアプリケーション要求
 - » タイムシェアリングプロセス動作
- アプリケーションごとの優先度調整は困難.

e.g.) RT /TSプロセス動作
 RT プロセスは実行後定期的に 2秒間 sleep
 - RTプロセス (赤線)
 - TSプロセス (青線)

RTプロセスが動作中,
 GUIからキーボード入力不可.



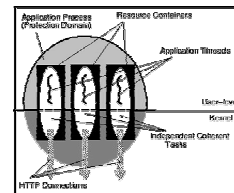
2005/09/30

TechnicalJamboree4

6

資源管理に関する関連研究・開発事例

- RK/Linux
 - 資源予約 : Resource Set, CPU, Memory, Network, disk bandwidth
 - Shuichi, Oikawa, R. Rajkumar, Portable, RK: A Portable Resource Kernel for Guaranteed and Enforced Timing Behavior, RTAS, June 02 - 04, 1999.
- Resource Container
 - アプリケーションスレッドが使用する資源保護
 - httpアクセス, CPU, memory, socket, etc.
 - Resource containers: A new facility for resource management in server systems (1999), Gaurav Banga, Peter Druschel, Jeffrey C. Mogul OSDI.
- CKRM
 - Class-based Kernel Resource Management (CKRM)
 - マネージャデーモン, クラス割り当て, 階層型スケジューリング
 - スループットに重点
 - CPU, Memory, Disk/I/O bandwidth, Inbound socket connections
 - Enterprise server 向けの高機能な資源管理サービス
 - Ottawa Linux Symposium July 2004.
- Linux が提供している資源管理機構
 - rlimit : 資源量の調整
 - 秒単位での対処
- 組み込み用途を考えると...
 - 高精度な応答性を目的とするものがない
 - 設定インターフェイスが複雑
 - 資源制約が大きい組み込みには規模が大きい
 - シンプルで軽量なものが必要



2005/09/30

TechnicalJamboree4

7

組み込み用途のQoS システムの実現

- 早大: 中島研究室での取り組み
 - アカウンティングシステム(CABI)の開発
- Emblix リソースマネジメントワーキンググループ
 - プロセスのQoS に関する標準化仕様の策定

2005/09/30

TechnicalJamboree4

9

組み込み Linux のスケジューリングの課題

TechnicalJamboree4
2005/09/30

Linux のリアルタイム性能

- “A Measurement-Based Analysis of the Real-Time Performance of Linux”
 - Luca Abeni , Ashvin Goel, Charles Krasic, Jim Snow, Jonathan Walpole, RTAS 2002.
- Identify major source of latency in the kernel with **the goal of providing real-time performance in a widely used general-purpose operating system.**
- Until recently, **Linux has not focused on time-sensitive applications,** which are characterized by **temporal constraints.**
 - The periodic execution, which is derived from the frame rate of an audio/video stream.
 - Require response in a short time to external events such as the arrival of a network packet.
- Temporal constraintsを満たすには、予測可能なスケジューリングが必要
- しかし、Linux は Latency の存在により、予測可能性が著しく低い

2005/09/30

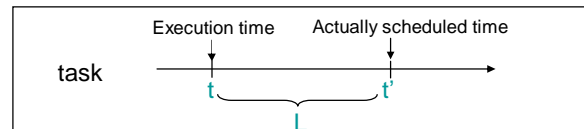
TechnicalJamboree4

11

Latency

• Definition of [OS Latency]

- Let τ be a task¹ belonging to a time-sensitive application that requires execution at time t , and let t' be the time at which τ is actually scheduled; we define the OS latency experienced by τ as $L = t' - t$.



- Two main causes of latency in the OS.
 - タイマの解像度
 - 非プリエンタブルセクションの存在
- ¹ここでの task は thread または process

2005/09/30

TechnicalUmboree4

12

タイマの解像度

• 周期的な tick 割り込み

- x86 では, PIT (Programmable Interrupt Timer) 周期 T^{tick} 10ms
 - プロセスの実行時間の計測など
 - 時間関係の API など利用される
- Tick Rate : HZ (frequency) は変更可能
 - e.g. Include/asm-i386/param.h
 - #define HZ 100 /* internal kernel time frequency */
 - e.g. Architecture (Frequency)

Architecture	Frequency (in Hertz)
alpha	1024
arm	100
i386	100
ppc	1000
sh	100 or 1000
sparc	100

2005/09/30

TechnicalUmboree4

13

タイマの解像度による遅延の縮小

- 周期の解像度を高くする事により
 - 遅延を縮小することができる
 - e.g. プロセスの残り実行可能時間が 2ms の場合
 - 解像度 1ms では、2ms で次のプロセスにスイッチするが、10ms の場合には、次のタイマ割り込みまで 8ms の遅延発生
- トレードオフ: 正確さ vs オーバーヘッド
 - 解像度高: 粒度が高いため、正確さは達成できる。しかし、
 - タイマハンドラ処理の比率の上昇
 - コンテキストスイッチのオーバーヘッド比率の上昇
 - » Kernel- 2.6, x86 はデフォルトで 1ms の周期
 - » 組み込みレベルのCPUでのオーバーヘッドの計測が必要
 - 解像度低: オーバーヘッドは低下、しかし最大遅延時間は増大
- 高解像度タイマ
 - 周期の解像度を高くする方法ではなく実現する
 - しかし、全てのアーキテクチャで利用可能ではない。

2005/09/30

TechnicalJamboree4

14

組み込み Linux のリアルタイム性能の向上

- プリエンプション遅延の縮小
 - プリエンプション改善のための様々なアプローチにより改善
 - Real-time benchmark results : <http://kpreempt.sourceforge.net/>
 - Low-Latency Linux kernel
 - Preemptable Kernel Patches
 - Preemptable Lock-Breaking
 - カーネルプリエンプションに関わる問題
 - 優先度逆転, デッドロックの発生
 - mutex や、優先度継承, 優先度上限などの方法により回避
- ただし,
 - 優先度の調整
 - 資源占有問題
 - 優先度の低いプロセスの応答性の改善
 に関してはまだ十分な解決策が考慮されていない。



これらの問題について考慮するためにはLinux のスケジューリングの構造を考慮する必要がある。

2005/09/30

TechnicalJamboree4

15

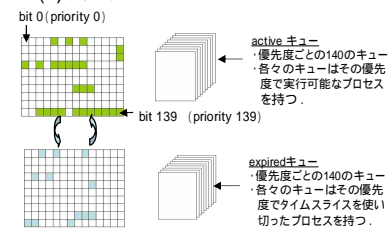
スケジューラの改善

- 予測可能性 (リアルタイム性能) の向上
 - Linux 2.4 までのカーネル
 - プロセスの選択 : 線形検索
 - 検索コスト : プロセス数の増加に比して検索のコストが増大
 - Jitter の発生 : キューが長い場合/短い場合でオーバーヘッドが異なる
 - O(1) スケジューラによる改善
 - Bitmap search による高速化
 - 各アーキテクチャの find-first-set instruction を利用
 - » e.g. x86, bsfl, ppc, cntlzw

- 優先度ごとに独立したキュー
- 優先度に対応した bitmap (140)

```
struct prio_array {
    /* number of tasks in the array */
    int nr_active;
    /* priority bitmap */
    unsigned long bitmap[BITMAP_SIZE];
    /* priority queues */
    struct list_head queue[MAX_PRIO];
}
```

・O(1) スケジューラ



2005/09/30

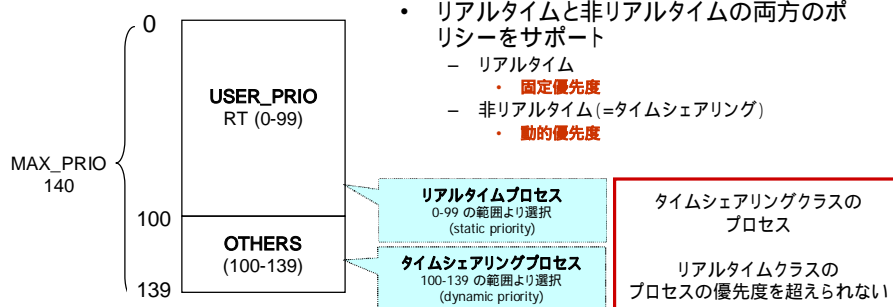
TechnicaUmboree4

17

Linux スケジューラの特徴

- Linux のスケジューリングクラス

タイプ	クラス	内容	優先度
リアルタイム	SCHED_FIFO	FIFO方式のリアルタイムプロセス	0-99
	SCHED_RR	ラウンドロビン方式のリアルタイムプロセス	
非リアルタイム	SCHED_OTHER	非リアルタイムプロセス	100-139



- リアルタイムと非リアルタイムの両方のポリシーをサポート
 - リアルタイム
 - 固定優先度
 - 非リアルタイム (=タイムシェアリング)
 - 動的優先度

2005/09/30

TechnicaUmboree4

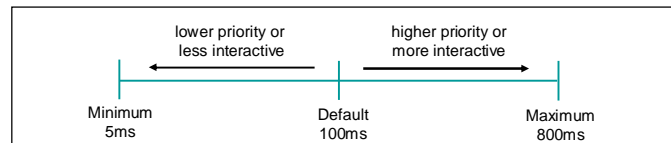
18

動的優先度クラス (1)

- Linux の動的優先度ポリシー

- 一定時間ごとに優先度を再計算
 - 実行時の振る舞いに基づき動的優先度決定
 - タイマ割り込み(10ms) 毎に, CPU利用時間, sleep 時間をカウント
 - 再計算時に, sleep 時間が長いプロセスの優先度をUP
 - Linux の実装:
 - 現在動作中のプロセスよりも待ちプロセスの優先度が高くなったら, need_resched (1)

- 再計算時のタイムスライスの割り当て



- I/O バウンドな処理に有利
 - イベント待ち (sleep) 時間が長い程優先度向上
 - 優先度が高い状態で待ち状態
 - イベントが生じた場合の応答性が向上

2005/09/30

TechnicalJamboree4

19

動的優先度クラス (2)

- Linuxのタイム動的優先度クラスは, Traditional UNIX Scheduling を踏襲

$$\begin{aligned} \text{CPU}_j(i) &= \text{CPU}_j(i-1) / 2 \\ P_j(i) &= \text{Base}_j + \text{CPU}_j(i) + \text{nice}_j \end{aligned}$$

$\text{CPU}_j(i)$: プロセスj がインターバルiの間のCPU利用率(時間).
 $P_j(i)$: インターバルi 開始時点でのプロセスj 優先度. 値が小さいほど優先度が高い.
 Base_j : プロセスj のベースとなる優先度
 nice_j : ユーザ指定の調整優先度.

CPU利用率(時間)が長いほど, P となり, 優先度は下がる.

- 動的優先度アルゴリズム

- 歴史的に Non-preemptible な方法で応答性を改善
 - Shortest Process Next (SPN), Shortest Remaining Time (SRT)
- 定期的(一定時間おき)にプリエンプションをかける
 - 応答性については限界
- タイマ割り込み
 - 優先度が現時点で実行しているものよりも高くなったらコンテキストスイッチ
 - タイマハンドラ終了時に need_resched のチェック

- しかし,

- リアルタイムプロセスが動作している場合は, 動的優先度クラスのプロセスはCPUを得られない 優先度クラスの上限の優先度は超えられないため

2005/09/30

TechnicalJamboree4

20

静的優先度クラス

- リアルタイムプロセス
 - 優先度キューの中の非リアルタイムクラスのプロセスとは異なる方法で扱われる
 - 静的な優先度のみを持つ。
 - リアルタイムの2つの区分
 - SCHED_FIFO
 - タイムスライスの割り当ては行われない
 - 処理が続く限り連続実行
 - ブロックされた場合
 - アンブロック時にはactive キューリストの中の同じ優先度キューに戻る。
 - SCHED_RR
 - タイムスライスの値は変更しない
 - タイムスライスの終了時に同or高優先度プロセスと交換
 - タイムスライスを使いきったとき、同じタイムスライス値を持つ優先度キューに戻る (expired キューに移行しない。)
- 動的優先度のプロセスの実行時間を定量的に残す (調整) のは面倒。
- e.g. マルチメディア
- O(1)スケジューラでは、
- active キューリストとexpired キューリストの間のスイッチは、リアルタイムタスクがないときにしか行われない。

2005/09/30

TechnicalJamboree4

21

スケジューリングの課題

- プロセス間のプリエンブションのパターン
 - RT: リアルタイムプロセス, TS: 非リアルタイム (タイムシェアリング) プロセス

		割り込まれる側	
		RT	TS
割り込む側	RT	パターン (RT RT) <ul style="list-style-type: none"> 優先度によるプリエンブション 優先度を適切に与えれば、応答性は改善 RM などのアルゴリズムの適用を考慮できる 一時的負荷の考慮が必要 	パターン (RT TS) <ul style="list-style-type: none"> 優先度によるプリエンブション 課題は と同じ
	TS	パターン (TS RT) <ul style="list-style-type: none"> 優先度によるプリエンブション不可 応答性が得られない 	パターン (TS TS) <ul style="list-style-type: none"> 動的優先度によるプリエンブション ブロック時間が長いプロセスの優先度が上昇 I/O バウンドな処理の応答性が確保

- 課題
 - 現状の Linux では
 - パターン のタイムシェアリングプロセスの応答性を上げる方法がない
 - リアルタイムプロセスが実行し続ける限り実行されない
 - Linux では優先度が低くても実行すべきプロセスが多数存在する
 - 例) XServer

2005/09/30

TechnicalJamboree4

22

応答性改善のためのアプローチ

- 2つのアプローチ
 - 優先度によるアプローチ
 - 非優先度によるアプローチ

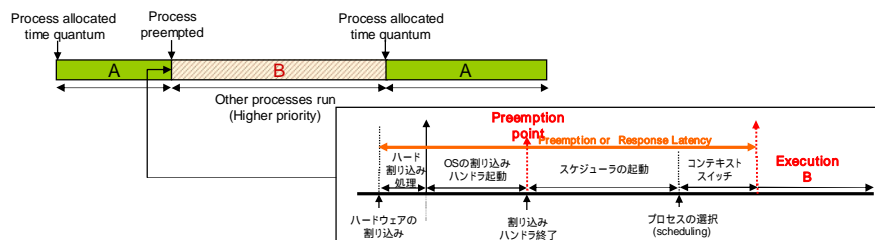
2005/09/30

TechnicalJamboree4

23

優先度によるアプローチ (プリエンプションモデル)

- Linux preemption patch
 - プリエンプションポイントの設置
 - 契機: 割り込みハンドラの戻り時
 - 優先度が高いプロセスが実行可能になっている場合にプリエンプション
 - need_resched(スケジューラ起動), preempt_count(non-preemptible section でない)
 - If need_resched (1) && preempt_count (0)
 - » スケジューラが呼ばれ、コンテキストスイッチ
 - Non-preemptible section : 割り込みハンドラ, スピンロックセクション等



A の優先度が最も (十分) 高い場合, B はプリエンプション不可 (予測可能) .
 = B の優先度により調整

2005/09/30

TechnicalJamboree4

24

優先度によるアプローチ

• 優先度の低いプロセスの優先度を上げる

- 優先度の低いプロセスの優先度を、割り込み先のプロセスより上げる
 - 優先度を上げる契機、下げる契機を考慮して優先度を変更
 - set_scheduler, sched_yield などの方法で対応が可能
 - アプリケーション側での考慮が必要
 - » 優先度を下げ忘れると、バグのもととなる

• 結果

- 優先度が低いプロセスの応答性は向上
 - 開発者が優先度を調整
 - 製作したアプリケーション、着目しているアプリケーション
 - » 対応可能
 - 但し、
 - 意図しないバグによる資源占有には対処不能
 - 優先度調整にかかる開発者の手間は増える一方

2005/09/30

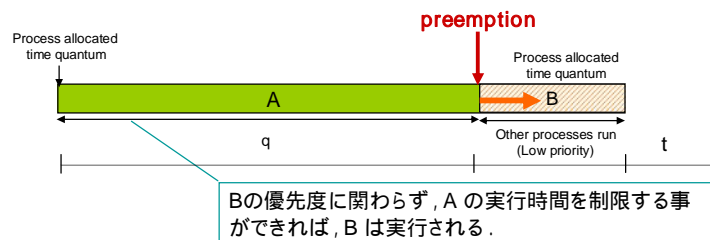
TechnicalJamboree4

25

非優先度プリエンプシヨンのアプローチ

• 強制的なプリエンプシヨンの

- ある周期内で、Aが利用できる最大実行時間を制限
 - e.g. $q = 70\text{ms}$, 周期 = 100ms (70%)
- 優先度の低い B が残りの CPU 時間を利用することができる



- 現在実行中のプロセス優先度に関係なくプリエンプシヨンを行う。
- 定量的に指定することで、資源占有を防ぐことができる。
 - プリエンプシヨポイント: 定量的に指定した相対時間 (比率)。
 - 優先度の低いプロセスの応答性の改善

2005/09/30

TechnicalJamboree4

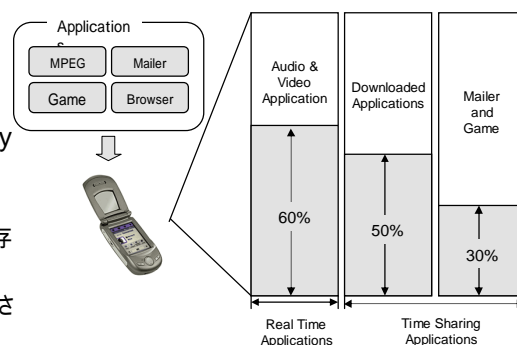
26

アカウンティングシステムの提案

TechnicalJamboree4
2005/09/30

要求仕様

- 組み込み用途のQoS システム
 - アプリケーション毎の実行比率制御を自由に行うことができる
- 要求仕様項目
 - Security
 - 資源占有の防止
 - Improving Responsibility
 - 応答性改善
 - Independency
 - スケジューラへの非依存
 - Accuracy
 - 組み込みに適した正確さ
 - Simplicity
 - 容易に利用可能



2005/09/30

TechnicalJamboree4

28

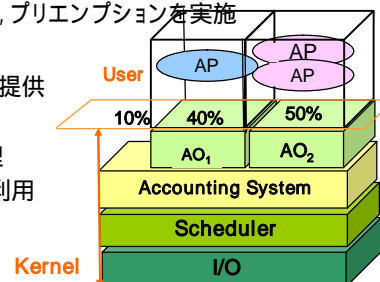
アカウンティングシステム

- CABI (CPU Accounting and Blocking Interfaces)

- CPU時間の監視 & プリエンプションシステム
- 非優先度アプローチ

- 特徴

- AO (Accounting Object) 単位による制御
 - AOごとにユーザは使用率(実行時間/周期)を設定
 - プロセスのCPU利用時間を監視し、プリエンプションを実施
- AOの制御を含むCPU資源管理機構を提供
 - 複数バインド可能(資源の共有)
 - AOによるプロセスグループの管理
 - HRT (High Resolution Timer) の利用



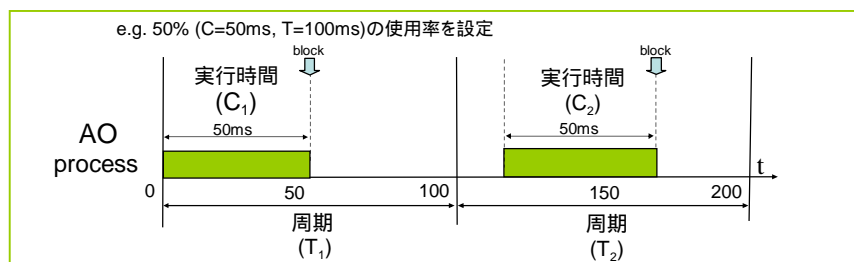
2005/09/30

TechnicalUmboree4

29

アカウンティングシステムのプリエンプションモデル

- 定量的に動作中のプロセスの実行時間を制限 .
 - T (周期: period time), C (実行時間: computation time) を設定 .
 - リアルタイムプロセス, 非リアルタイムプロセス両方に適用可能
 - 実行時間の終了では, タイマ割り込みハンドラ (add_timer()) を利用
 - タイマ割り込みの優先度 (ハードウェア割り込みとしては最優先される)
- プロセスのCPU使用率 $C/T(\%)$ に応じてプリエンプションを行う .
 - AO (Accounting Object) にプロセス, プロセスグループをバインドする .
 - AO は単数, 複数のプロセスのCPU使用率を管理する .
 - プロセスの優先度に関係なく, プリエンプションを行うことができる .



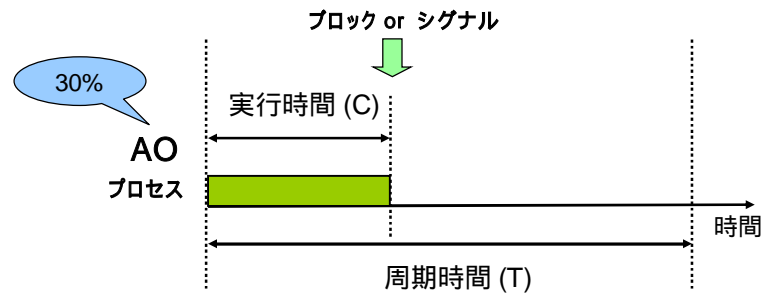
2005/09/30

TechnicalUmboree4

30

実行時間の管理

- AO x 1, プロセス x 1
 - 実行時間(C)の終了時に, ブロック又はシグナルを受ける
e.g. C=30ms, T=100ms



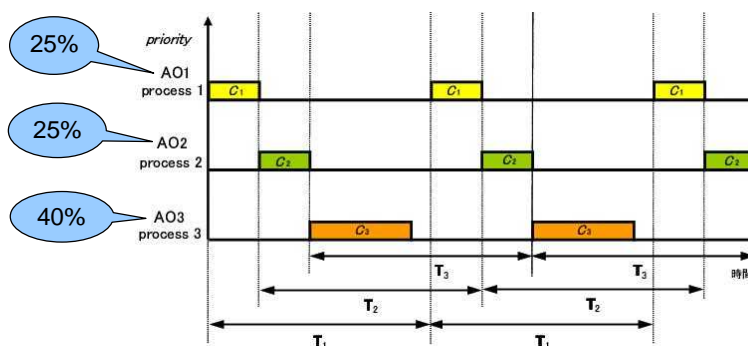
2005/09/30

TechnicaUmboree4

31

実行時間管理 (複数プロセス)

- AO x 3, プロセス x 3 (AO_n , process_n)
 - それぞれの AO_n にプロセスをバインド
e.g) C₁, C₂=25ms, C₃=40ms, T₁, T₂, T₃=100ms



2005/09/30

TechnicaUmboree4

32

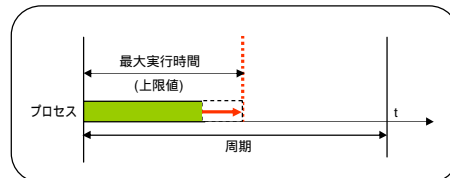
アカウントティングシステムによる時間制御

• 定量的な制限と保証

- Linux スケジューラのポリシーとの兼ね合い
 - スケジューリングクラスによって制限 (上限値) か保証 (下限値) か異なる利用が可能

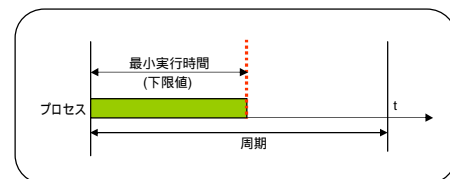
• 制限 (上限値)

- 指定した以上の時間を使用しない
- 対象: 非リアルタイムプロセス
 - デッドラインが存在しない
 - 優先度が高いプロセスにリソース譲る



• 保証 (下限値)

- 指定した時間を保証する
- 対象: リアルタイムプロセス
 - 優先度の考慮が必要
 - 十分な優先度があれば保証可能
 - » 資源予約



2005/09/30

TechnicaUmboree4

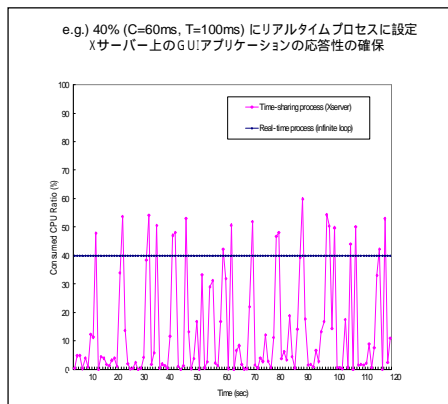
33

優先度が低いプロセスの応答性の改善

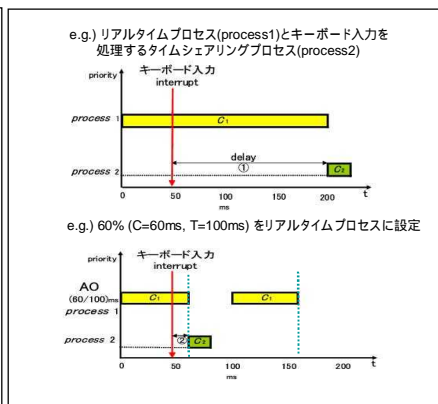
• タイムシェアリングプロセス

- 優先度の高いリアルタイムプロセス (クラス) の実行を定量的に制限

Xサーバ上のGUIアプリケーションの例



キーボード入力処理の例



2005/09/30

TechnicaUmboree4

34

リアルタイム性の保証と応答性の改善

- 一時負荷時
 - RMA (Rate Monotonic Algorithm)
 - 重要度の高いプロセス(デッドラインが長い)からデッドラインをミス
 - アカウンティングシステム(高解像度タイマ)を用いた周期変更
 - RMA をベースとし, 重要度を反映した最適スケジューリングの実現
 - 組み込みの要求(細粒度)に対応可能
- アカウンティングシステムによる制御時間の変更
 - アカウンティングシステムにより, 実行時間/周期を変更
 - 重要度が高いプロセスの優先度を高める
 - 重要度の高いプロセス
 - » 音声などを処理するストリーミング処理プロセス
 - » ソフトリアルタイム下で優先度を高くして実行したい場合
 - » RMA では, 処理時間が長いため, 優先度が低くなる
 - 重要度を反映した周期の変更
 - $P_c > P_n$ 時間, $P_{c,max} > P_{n,min}$ [c (critical), n (non-critical)]
 - 定数(k) による周期変更: critical処理プロセスの最大周期が大きき場合には縮小

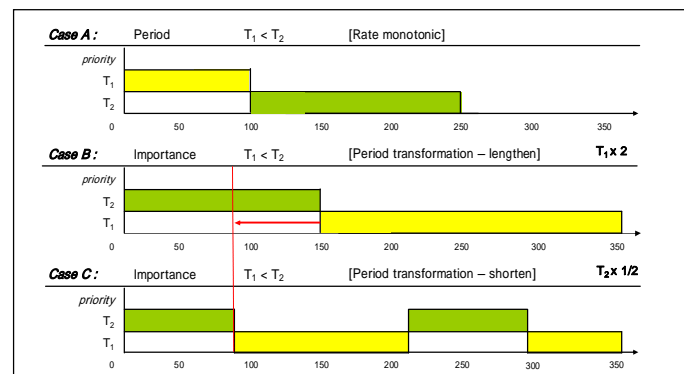
2005/09/30

TechnicaJamboree4

35

重要度を考慮したリアルタイム性向上と応答性の改善

- 重要度を考慮したリアルタイム性と応答性の改善の例
 - 2つのプロセスがそれぞれ, $T_1 = 100ms$, $T_2 = 150ms$ の周期時間を要求
 - RMAでは, 周期時間が短い T_1 が先に実行される (Case A)
 - 周期変更の2パターン
 - Case B 重要度の高いプロセスの周期を k 倍
 - Case C 重要度の低いプロセスの周期を $1/k$ 倍
 - Case C の方が応答性が高い。 応答性とリアルタイム性を考慮したスケジューリングとなる,



2005/09/30

TechnicaJamboree4

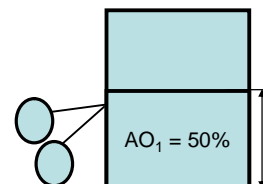
36

CABI (CPU Accounting & Blocking Interfaces) 技術内容

TechnicalJamboree4
2005/09/30

アカウントティングシステムの利用

- 管理オブジェクト (AO: Accounting Object)
 - プロセスは管理オブジェクトにバインドして制限する
- システムコールにより制御
 - 作成
 - 制限時間を設定
 - C= 50ms , T= 100ms
 - 100ms 中 50 % にあたる 50ms までで制限される
 - バインド
 - CPU時間を制限したいプロセスを AO にバインド
 - アンバインド
 - プロセスをAOの管理対象から外す
 - 削除
 - 管理オブジェクトの削除
 - 設定値の参照, 変更



2005/09/30

TechnicalJamboree4

38

アカウントシステムが提供するAPI

- AO 構造体 (cabi_account)
 - 周期 (T), 実行時間(C)の値を設定
 - 終了動作(block/signal), プロセス ID など
 - AO毎に識別子(object_id) を持つ

• システムコール API

システムコールインターフェイス	内容
int cabi_account_create (obj_id, &object_attr)	オブジェクトの作成
int cabi_account_destroy (obj_id)	オブジェクトの削除
int cabi_account_pid_bind (obj_id, &object_attr)	オブジェクトにプロセスをバインド
int cabi_account_pgid_bind (obj_id, &object_attr)	オブジェクトにプロセスグループをバインド
int cabi_account_set(obj_id, &object_attr)	オブジェクトの設定値の変更
int cabi_account_get(obj_id, &object_attr)	オブジェクトの設定値の取得

2005/09/30

TechnicalJamboree4

39

データ構造

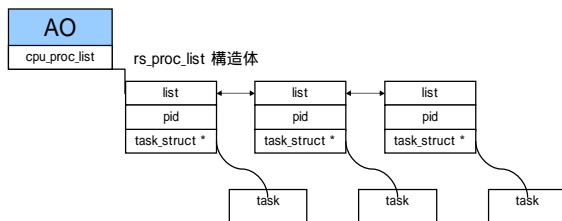
パラメータ例:

AOステータス : AO は次の 4つの状態(ステータス)を持つ。
 CABI_IS_NULL AO は動作していない
 CABI_IS_RUNNING AO は実行中
 CABI_IS_DEPLETED AO は空である
 CABI_IS_OVERLOAD AO はオーバーロードである

AO パラメータ : パラメータは次の3つの設定ができる。
 - 終了動作(terminate action) (cabi_terminate_action 共同体)
 CABI_TERM_NONE 何もしない
 CABI_TERM_BLOCK ブロック
 CABI_TERM_SIGNAL シグナル

シグナル時のプロセスIDとシグナル送信先 (cabi_signal構造体)
 pid_t pid プロセスID
 int sig シグナル番号

AO にバインドするプロセスのタイプ (cabi_bind_porc 共同体)
 BIND_IDLE_PROCS アイドルプロセス
 BIND_NORMAL_PORC 通常のプロセスAO オブジェクト ID



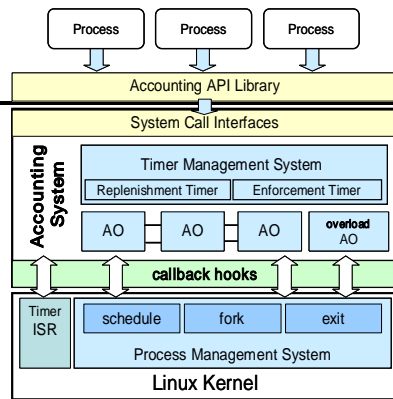
AOステータス
AOパラメータ
終了動作
シグナルの詳細情報
バインドするプロセスのタイプ
AO オブジェクトID
オーバーロードフラグ
AO プロセスリスト
AO の /proc のエントリポイント
周期時間(T)を管理するタイマ
周期時間(T)内の使用済時間 (ticks)
周期時間(T)内の残り時間 (ticks)
前回の使用済みの時間 (ticks)
総実行時間 (ticks)
スケジュールされた総(使用時間/周期時間)の合計
AOの実行時間
AOの周期時間
AOの実行時間 (ticks)
AOの周期時間 (ticks)
AOの実行容量
AOの終了処理状態
ブロック実行カウンタ
シグナル送信カウンタ

2005/09/30

TechnicalJamboree4

40

システムアーキテクチャ



1. Accounting API Library
2. System Call Interfaces
3. Process Management Function
 - schedule_hook
 - fork_hook
 - exit_hook
4. Timer Management Function
 - Replenish Timer
 - Enforcement Timer
5. AO Management Function
 - AO state transition
 - NULL
 - RUNNING
 - DEPLETED
 - OVERLOAD

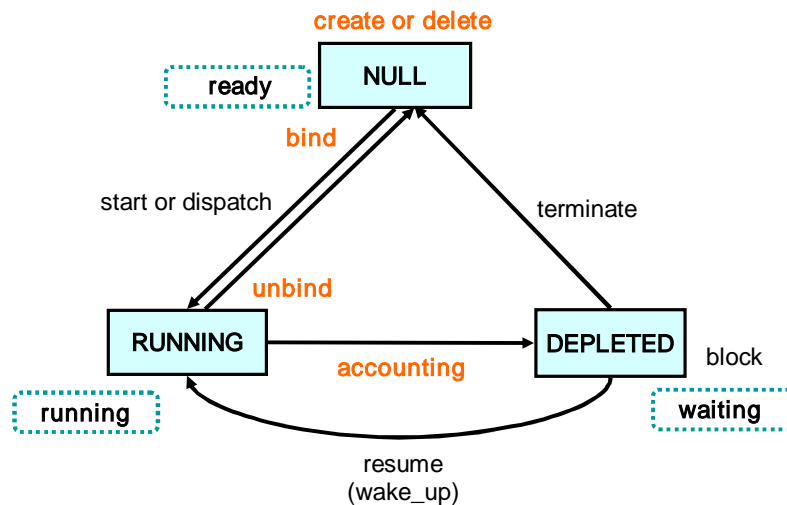
2005/09/30

TechnicaUmboree4

41

AOの状態遷移

- ・ AO は, NULL, RUNNING, DEPLETED (期限切れ) の3つの状態を持つ.



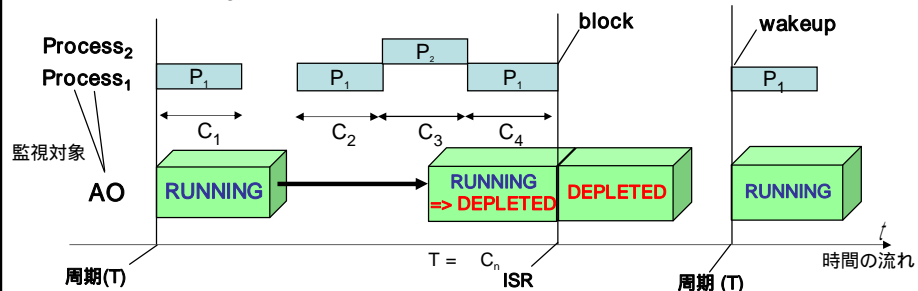
2005/09/30

TechnicaUmboree4

42

AO の状態遷移によるプロセスの制御

- Data + Time + Status
 - **Data** : オブジェクトのデータ構造
 - **Time** : オブジェクトによる時間管理
 - **Status** : オブジェクトの状態
- オブジェクトの状態遷移によりプロセスを制御
 - オブジェクトによるプロセス制御モデル
 - OS の提供するタイマ, 割り込み機能により実現
 - e.g) AO にバインドされている process1, process2, プロセスの実行時間(C), 周期(T)



2005/09/30

TechnicalJamboree4

43

プロセス管理システム

- Kernel 本体のプロセス管理部分への最小限のフックを設置
 - schedule_hook
 - コンテキストスイッチの直前
 - プロセスの実行の開始/終了時間を取得
 - fork_hook
 - 子プロセスの作成後
 - 親がAOにバインドされている場合にはその ID を引き継ぐ
 - exit_hook
 - 終了処理の直後
 - AO にバインドされている場合にはアンバインドする
- /proc ファイルシステム
 - 情報の表示
 - アカウンティングしているプロセスID, ブロック/シグナルなどのイベント
 - 周期時間, 実行時間
 - 今後 /proc (sysfs) 経由でのパラメータの設定など考慮

2005/09/30

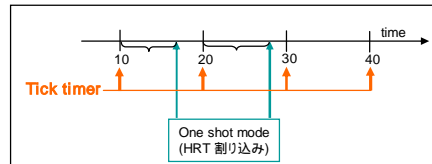
TechnicalJamboree4

44

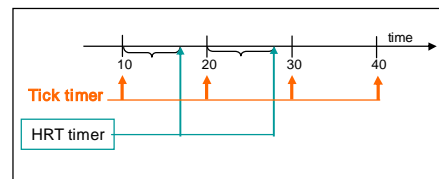
高解像度タイマの利用

- 高解像度タイマは、タイマ割り込みのオーバーヘッドを減らし、細粒度の時間を提供する。
 - High-resolution timer patch (<http://sourceforge.net/projects/high-res-timers/>)
- μ秒, nano秒単位のAPIを提供
 - For application
 - e.g. alarm(), setitimer()/getitimer(), sleep()/usleep(), nanosleep()
 - For kernel
 - e.g. add_timer()/del_timer(), mod_timer()
 - subjiffies
 - サブジッフィを設定できる
- アーキテクチャ毎に実装は異なる
 - x86 :
 - APIC (Advanced Programmable Interrupt Controller)
 - SH :
 - CPU内臓タイマデバイス 0,1 (ARM, MIPS)

• x86



• SH, ARM, MIPS



2005/09/30

TechnicalJamboree4

45

アカウンティングシステムの適用 (1)

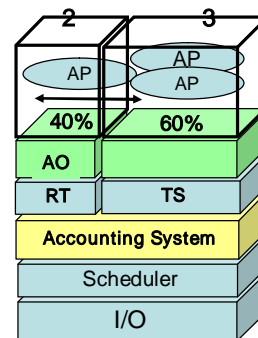
- クラス別アカウンティング
 - アカウンティングシステムの問題点
 - リアルタイムプロセス制御のミスによる資源占有
 - 意図せざる資源占有
 - スケジューリングクラスごとの資源を分ける
 - リアルタイムスケジューラ (RT), タイムシェアリングスケジューラ (TS)
 - ウェイト (比率) を指定
 - 2, 3 : wt=2 (2/5), wt=3 (3/5)
 - 比率はウェイトで指定
 - e.g) リアルタイムクラス、タイムシェアリングクラスの比率 (2:3)
40% vs 60%

$$B_i = \frac{r_i}{\sum_{j=1}^n r_j} \times B$$

B : 全体の資源量
 B_i : クラス i が受取る資源量
 r_i : i 番目のウェイト

– メリット

- 100% で調整を行うため、確実にクラスに割り当てられた資源が保護される



2005/09/30

TechnicalJamboree4

46

アカウンティングシステムの適用 (2)

• オーバード監視機能

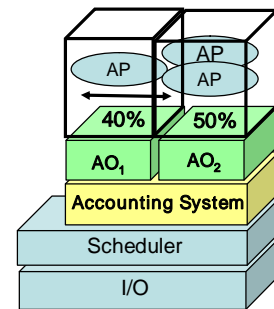
- システムの負荷が高まり, 不安定な状態
 - RTプロセス: デッドラインをミス, TSプロセス: 実行遅延
 - アカウンティングシステムを用いたシステム設計

- オーバード通知機能

- 指定した値を超えるとプロセスに通知
 - idle プロセスを AO にバインド.
 - » idle プロセスの監視
 - » システム全体の使用率と逆の比率で動作
 - 全体が指定した比率(%) を超えると通知.

• API

- overload 用の AO の作成, 削除
 - overload_create (&object_attr, pid)
 - overload_destroy (void)



2005/09/30

TechnicalJamboree4

47

CABI の評価 & 今後の課題

TechnicalJamboree4

2005/09/30

評価

- 試験環境
 - CPU Celeron 300MHz
 - MEMORY 128MB
- 方法
 1. 基本操作のコスト
 - 基本 API のユーザモードからの呼び出し
 2. 応答性の評価
 - TS/RTプロセスの動作
 3. 複数プロセスのバインド評価
 - 複数プロセスが起動するマルチメディアアプリケーションの実行
 4. 精度の向上
 - HRT の有無
 5. システムの保護
 - ハッキングプログラムによる調査
 6. クラス別アカウンティングの評価
 - クラス別アカウンティングの有無

2005/09/30

TechnicaUmboree4

49

評価 (1) 基本操作のコスト

- 基本的な API の実行時間(ユーザ<->カーネル)

API	User (μ sec)
create	39.3
bind	21.1
unbind	6.2
destroy	10.8
set	10.4
get	2.4

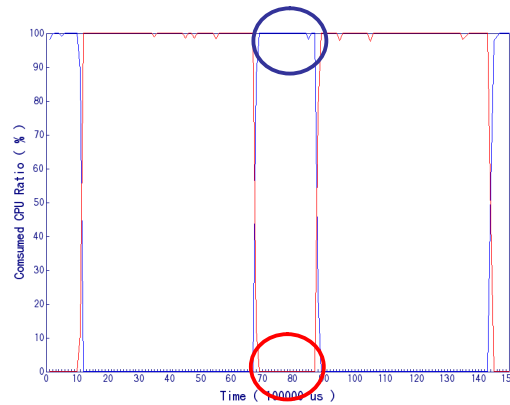
2005/09/30

TechnicaUmboree4

50

評価 (2)-1 アカウンティングシステムによる制限

- アカウンティングシステムを使用しない場合
 - — RTプロセス — TS プロセス
 - RT プロセスは実行後, 定期的に 2秒間 sleep



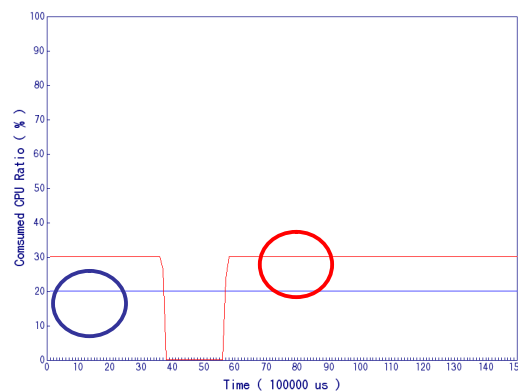
2005/09/30

TechnicaUmboree4

51

評価 (2)-2 アカウンティングシステムによる制限

- アカウンティングシステムを使用する場合
 - — RT 30% (C=30ms,T=100ms) — TS 20 % (C=20ms,T=100ms)
 - TS プロセスは実行時間を得られる



2005/09/30

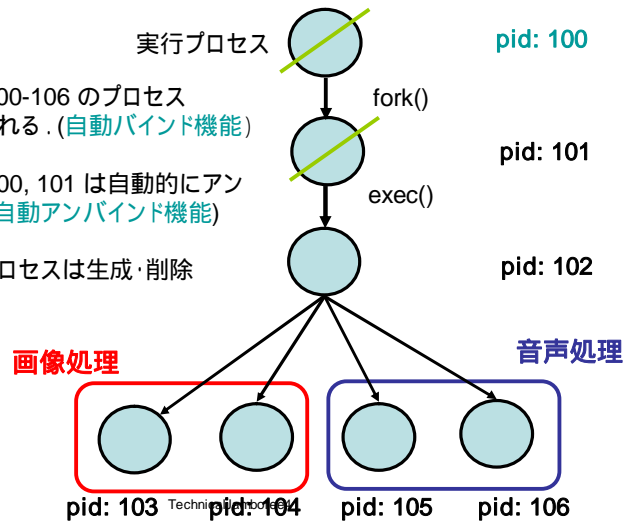
TechnicaUmboree4

52

評価 (3) 複数プロセスのバインドの評価

• plaympeg の実行処理

- CABI では pid 100-106 のプロセス全てがバインドされる。(自動バインド機能)
- 死んだプロセス100, 101 は自動的にアンバインドされる。(自動アンバインド機能)
- 一回転ごとに, プロセスは生成・削除される



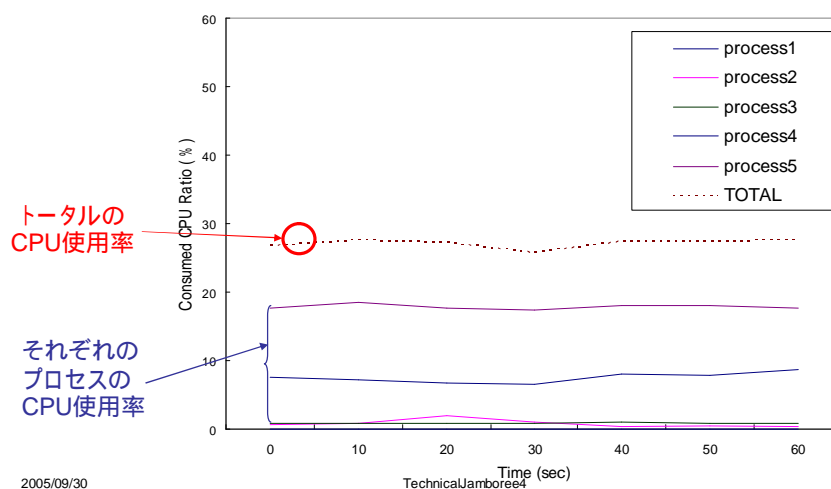
2005/09/30

TechnicaJamboree4

53

評価 (3) 複数プロセスのバインドの評価

- AO 30% (C=30ms, T=100ms)
 - 複数プロセスのトータルのCPU使用率



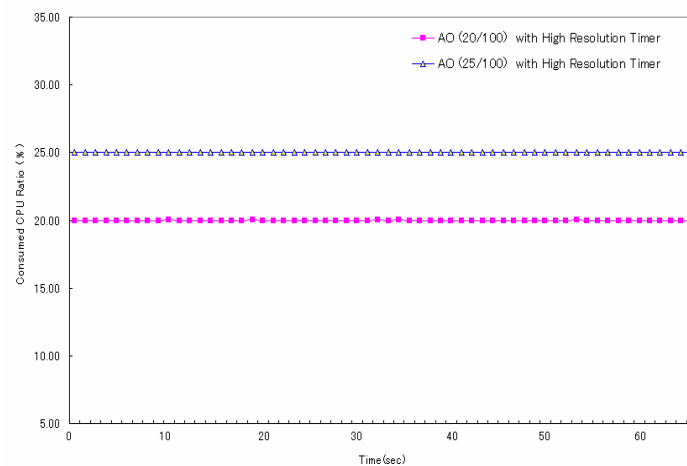
2005/09/30

TechnicaJamboree4

54

評価 (4) 精度の向上 1

- High Resolution Timer がある場合
 - AO₁ (20/100ms) 20 % と AO₂ (25/100ms) 25 % の検証結果

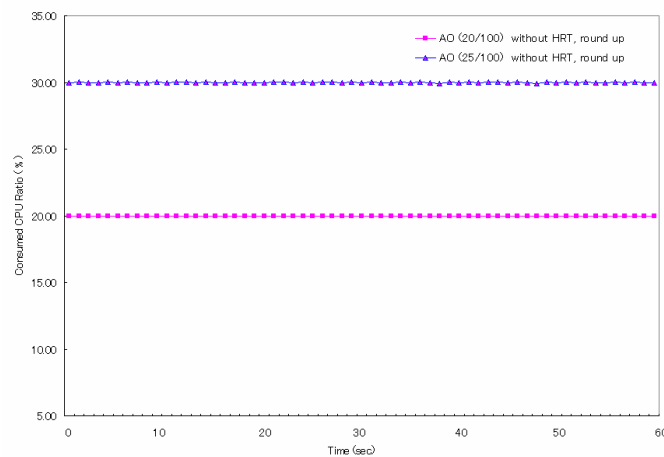


2005/09/30

55

評価 (4) 精度の向上 2

- High Resolution Timer が無い場合
 - AO₁ (20/100ms) 20 % と AO₂ (25/100ms) 25 % の検証結果
 - 切り上げがない場合, 両方とも 20% -> 切り上げ後 20%, 30%

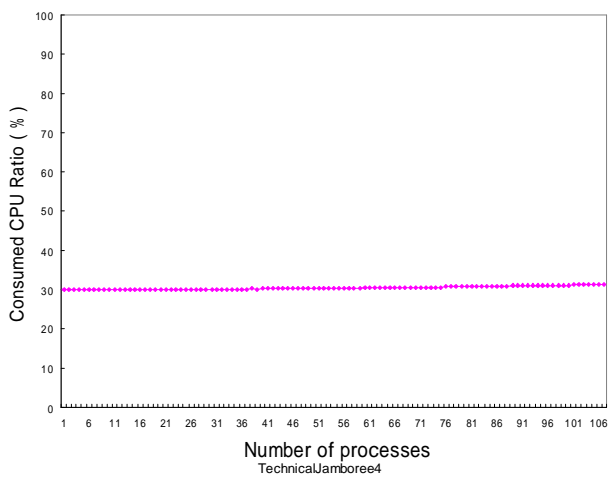


2005/09/30

56

評価 (5) システムの保護

- fork() によるハッキングプログラムによる検証
 - AO (30/100)ms に子プロセスを連続的に生成するプログラムをバインド
 - 50個 (30.27%), 100個(31.32%)

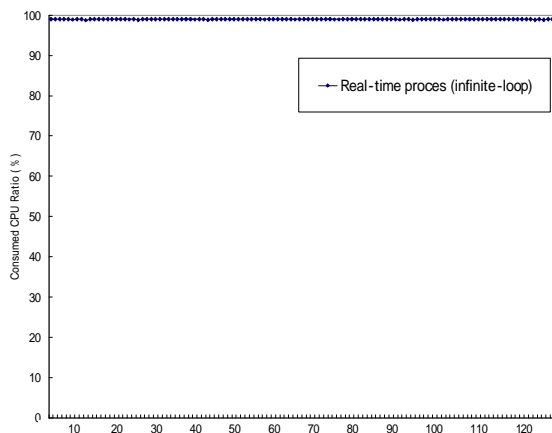


2005/09/30

57

評価 (6)-1 クラス別アカウンティングの検証

- クラス別アカウンティングを行わない場合
 - RT クラスでのプログラムの実行
 - RT プロセスによる 100% の資源占有が生じる .

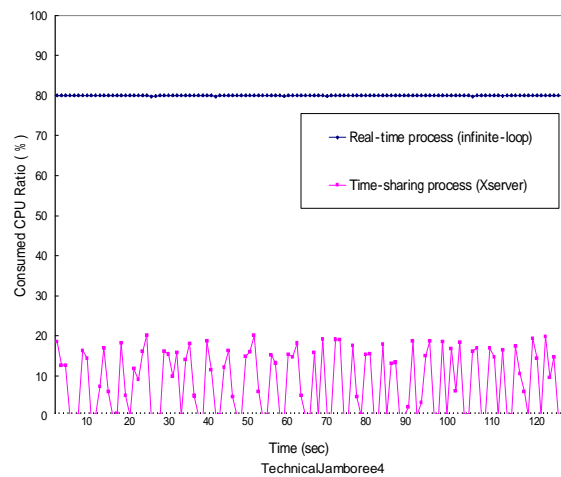


2005/09/30

59

評価 (6)-2 クラス別アカウンティングの検証

- クラス別アカウンティングによる制限が有る場合
 - TS クラス: 1, RT クラス: 4 の検証結果
 - この場合, それぞれ TSクラス(20%), RTクラス(80%) のCPUに制限される



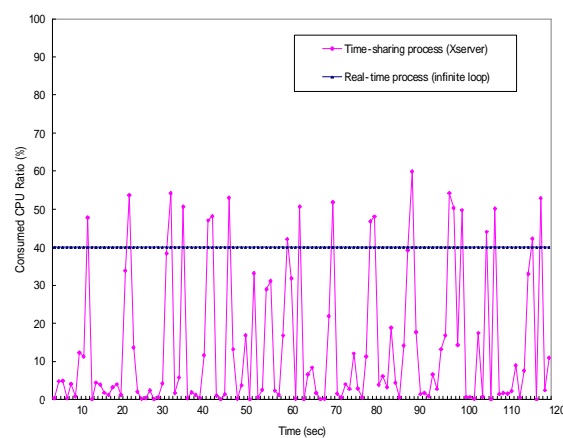
2005/09/30

TechnicalJamboree4

60

評価 (6)-3 クラス別アカウンティングの検証

- クラス別アカウンティングによる制限が有る場合
 - TS クラス: 3, RT クラス: 2 の検証結果
 - この場合, それぞれ TSクラス(60%), RTクラス(40%) に制限される



2005/09/30

TechnicalJamboree4

61

リファレンス実装の公開

- ppc, sh, mips, arm, x86へ対応
- Linux 2.6 対応
- Sourceforge にて開発バージョン公開
 - <http://www.sourceforge.net/cabi/>
 - <http://sourceforge.jp/>
- Emblix
 - <http://www.emblix.org>

SourceForge.jp project page for 'CABI: CPU Accounting & Blocking Interface'. The page shows a table of releases with columns for version, platform, size, download count, and date.

バージョン	プラットフォーム	サイズ	ダウンロード数	日付
1.0.0-CABI-PPC	PPC	14.9 KB	5	2005-09-13 15:11
1.0.0-CABI-ARM	ARM	14.9 KB	1	2005-09-13 14:55
1.0.0-CABI-MIPS	MIPS	14.9 KB	1	2005-09-13 14:55
1.0.0-CABI-SH	SH	14.9 KB	1	2005-09-13 14:55
1.0.0-CABI-X86	X86	14.9 KB	1	2005-09-13 14:55

Emblix website showing the '技術ドキュメント' (Technical Documents) section. It includes a table of releases with columns for version, platform, size, and date.

バージョン	プラットフォーム	サイズ	ダウンロード数	日付
1.0.0-CABI-PPC	PPC	14.9 KB	5	2005-09-13 15:11
1.0.0-CABI-ARM	ARM	14.9 KB	1	2005-09-13 14:55
1.0.0-CABI-MIPS	MIPS	14.9 KB	1	2005-09-13 14:55
1.0.0-CABI-SH	SH	14.9 KB	1	2005-09-13 14:55
1.0.0-CABI-X86	X86	14.9 KB	1	2005-09-13 14:55

2005/09/30

TechnicaUmboree4

63

Kernel-2.6対応について

- 経緯と現状
 - kernel-2.4.20版をベースに移植を開始 (2005/06 ~)
 - 現在、kernel-2.6.13.2ベースに移植作業を実施中 (完成度: 版)
 - 対応状況
 - 各種カーネルAPI変換業終了
 - 基本動作確認中(AO生成、プロセスへのバインド)
 - 対応アーキテクチャ
 - IA32
 - ARM
 - PPC
 - SH(未テスト)
- その他各種組込みCPU向けに対応作業実施中。

2005/09/30

TechnicaUmboree4

64

Emblix リソースマネジメント WG による 標準化への取り組み

TechnicalJamboree4

2005/09/30

リソースマネジメントワーキンググループ

- 発足 : 2004年6月
- 目的 :
 - Linux上のQoSを支援するための標準API の決定
 - 標準化の仕様書の策定
 - Emblix 標準化指標に準拠したQoSシステムの推奨と推進
- 参加 :
 - モンタピスタソフトウェア ジャパン(株), 兵庫県立大学, ウェブソフト・インターナショナル(株), 岩崎通信機(株), ソリッド(株), (有)マイト, (株)アックス, ヤマハ(株), 三菱電機(株), 名古屋大学, (株)KDDI研究所, 富士通(株), 早稲田大学
- 活動状況 :
 - プロセス QoS 制御に関する仕様の策定
 - 9月, 合意事項, 討論事項の整理
 - 12月(年内) 標準化仕様書をリリース目標

2005/09/30

TechnicalJamboree4

68

CPU資源制御の標準化

検討事項など

適用対象	<ul style="list-style-type: none"> ・ プロセス ・ プロセスグループ ・ スレッド
制御対象	<ul style="list-style-type: none"> ・ CPU時間 ・ 周期については、実時間(絶対時間)で定義する。 ・ 実行時間については、厳密な定義を定めない。 ・ 時間の単位は実装に依存する。(ただし、単位を知る手段は定義する)
制御内容	<ul style="list-style-type: none"> ・ 実行可能状態での制限 - 実行可能なプロセスが他にない等、 ・ 上限(制限) ・ 下限(保証)

・ CPU資源制御システムの使用手順

- 対象の作成
- 制御値の適用
- 制御値の変更
- 制御値の参照
- 削除

・ グループ化手法

- 継承
- GUID の参照

・ 実現方法

- /proc 経由
(適用タイミング、セキュリティを考慮)
- システムコール
(アーキテクチャ依存部が増えるのでデメリットを考慮)
- ioctl

・ 時間精度

- ・ オプション、実装依存とする。
- ・ 単位は μ sec, ms,

2005/09/30

TechnicalJamboree4

69

まとめ

- Linux の資源管理について

1. Linux の課題

2. 取り組み事例

- ・ 組み込み Linux 向け資源管理システム(CABI)
- ・ Emblix WG でのプロセスQoSに関する標準化の取り組み

について述べました。

ご意見、ご質問などありましたら、宜しくお願い致します。

2005/09/30

TechnicalJamboree4

71